DESIGN AND TESTING OF A GENERALIZED REDUCED GRADIENT CODE FOR NONLINEAR OPTIMIZATION

Leon S. Lasdon, et al

Case Western Reserve University

Prepared for:

Office of Naval Research

March 1975

DESIGN AND TESTING OF A GENERALIZED REDUCED GRADIENT CODE FOR NONLINEAR OPTIMIZATION

by

Leon S. Lasdon
Allan D. Waren
Arvind Jain
Margery W. Ratner

This report was prepared as part of the activities of the Department of Operations Research, School of Management, Case Western Reserve University, partially supported under Contract N00014 - 75 - C - 0240 with the office of Naval Research. Reproduction in whole or part is permitted for any purpose by the United States Goverment.

Technical Memorandum No. 353

March 1975

Index

1. Introduction

Generalized Reduced Gradient (GRG) Methods are algorithms for solving nonlinear programs of general structure. An earlier paper [1] discussed the basic principles of GRG and presented the preliminary design of a GRG computer code. This paper describes a modified version of that initial design, including the experiences that led to the modifications. This paper also is intended to serve as partial system documentation. The code is compared computationally with an interior penalty function code, and anticipated future work on the algorithm is outlined.

2. Brief Description of Generalized Reduced Gradient Algorithms

Generalized Reduced Gradient (GRG) Algorithms solve nonlinear programs of the form

$$\begin{aligned} &\text{minimize} && g_{M+1}(X) \\ &\text{subject to} && g_i(X) = 0, \quad i=1, \text{NEQ} \\ & && 0 \le g_i(X) \le UB(N+i), \quad i=NEQ+1, M \\ & && LB(i) \le X_i \le UB(i), \quad i=1, N \end{aligned} \tag{1}$$

where X is a vector of N variables. NEQ, the number of equality constraints, may be zero. The functions g_i are assumed differentiable.

There are many possible GRG algorithms. Their underlying concepts are described in references [1] - [3]. This paper briefly describes the version currently implemented in our code.

The user submits the problem in the above form. It is converted to the following equality form by adding slack variables $X_{N+1}, \ldots, X_{N+M}$:

$$\begin{aligned} &\text{minimize} && g_{M+1}(X) \\ &\text{subject to} && g_i(X) - X_{N+i} = 0, \quad i=1, M \\ & && LB(i) \le X_i \le UB(i), \quad i=1, N+M \\ &\text{where} \\ & && LB(i) = UB(i) = 0, \quad i=N+1, N+NEQ \\ & && LB(i) = 0 \quad i=N+NEQ+1, N+M \end{aligned} \tag{2}$$

These last two equations are the bounds for the slack variables. The variables $X_1, \ldots, X_N$ will be called "natural" variables.

Let $\bar{X}$ satisfy the constraints of (1), and assume that NB of the g_i constraints are binding (i.e. hold as equalities) at $\bar{X}$. A constraint g_i is taken as binding

if $\quad |g_i - UB(N+i)| < EPNEWT$

or $\quad |g_i - LB(N+i)| < EPNEWT$

i.e. if it is within EPNEWT of one of its bounds. The tolerance EPNEWT is one of the most critical parameters in the code. It can be set by the user, and has a default value of 10^{-4}.

GRG uses the NB binding constraint equations to solve for NB of the natural variables, called the basic variables, in terms of the remaining N-NB natural variables and the NB slacks associated with the binding constraints. These N variables are called nonbasic. Let y be the vector of NB basic variables and x the vector of N nonbasic variables, with their values corresponding to $\bar{X}$ denoted by $(\bar{y}, \bar{x})$. Then the binding constraints can be written.

$$g(y,x) = 0 \tag{3}$$

where g is the vector of NB binding constraint functions.* The basic variables must be selected so that the NB by NB basis matrix

$$B = (\partial g_i / \partial y_j)$$

is nonsingular at $\bar{X}$. Then the binding constraints (3) may be solved (conceptually at least) for y in terms of x yielding a function y(x), valid for all (y,x) sufficiently near $(\bar{y},\bar{x})$. This reduces the objective to a

*The definitions of g are extended here to include the slacks.

function of x only

$$g_{M+1}(y(x),x) = F(x) \tag{4}$$

and reduces the original problem (at least in the neighborhood of $(\bar{y},\bar{x})$), to a simpler reduced problem

minimize $F(x)$

subject to $\ell \leq x \leq u$

where ℓ and u are the bound vectors for x. The function F(x) is called the reduced objective and its gradient, $\nabla F(x)$, the reduced gradient.

This GRG code solves the original problem (1) by solving (perhaps only partially) a sequence of reduced problems. The reduced problems are solved by a gradient method. At a given iteration with nonbasic variables $\bar{x}$ and basic variables $\bar{y}$, B^{-1} is computed, and $\nabla F(\bar{x})$ is evaluated as follows:

$$u = (\partial g_{M+1}/\partial y)^T B^{-1}$$

$$\partial F/\partial x_k = \partial g_{M+1}/\partial x_k - u\, \partial g/\partial x_k$$

A search direction $\bar{d}$ is formed from $\nabla F(\bar{x})$ and a one dimensional search is initiated, whose goal is to solve the problem

$$\underset{\alpha > 0}{\text{minimize}} \quad F(\bar{x} + \alpha\bar{d}).$$

This minimization is done only approximately and may be terminated for a variety of reasons (see section 5). It is accomplished by choosing a sequence of positive values $\{\alpha_1,\alpha_2,\ldots\}$ for α. These are generated by subroutine DMINRG, described in section 5. For each value α_i, $F(\bar{x}+\alpha_i\bar{d})$ must be evaluated. By (4), this is equal to $g_{M+1}(y(\bar{x}+\alpha_i\bar{d}), \bar{x}+\alpha_i\bar{d})$, so the basic

variables $y(\bar{x}+\alpha_i\bar{d})$ must be determined. These satisfy the system of equations

$$g(y, \bar{x}+\alpha_i\bar{d}) = 0$$

This system is solved by a variant of Newtons method. If Newtons method converges, and no constraints are violated at the solution, a new α value is selected and the one dimensional search process continues. If any g_i constraints or any bounds on basic variables y are violated, the code determines a new α value such that at least one such new constraint or variable is at its bounds and all others are satisfied. If certain conditions are met (see description of subroutine DMINRG, section 5), the new constraint is added to the set of binding constraints, the one dimensional search is terminated, and solution of a new reduced problem begins.

3. System Overview

The GRG code described here is composed of a main program and a number of subroutines. It is written in FORTRAN IV and is currently operative on a UNIVAC 1108 at Case Western Reserve University and an IBM 370-145 at Cleveland State University. It uses double precision arithmetic.

System input is described in the user documentation. The only user supplied subroutine required is GCOMP, which computes the functions g_i for given X. The code requires first derivatives of the functions g_i, but these may be computed by a system subroutine PARSH using finite difference approximations. Alternatively, the user may supply a subroutine PARSH which computes first derivatives by analytic formulas or other means.

The code operates in two phases. If the initial vector X does not satisfy one or more of the g_i constraints, phase I finds an initial feasible point or determines that there is none. This is done by minimizing a phase I

objective function, which is the sum of the constraint violations. Phase II starts with an initial feasible point and attempts to minimize the user-supplied objective g_{M+1}. As with any other NLP algorithm any solution found may be only a local rather than a global minimum. In phase I, this means that a feasible point may not be located even if one exists. A popular procedure, if local optima appear to be a problem, is to try a variety of starting points. If the same final point is obtained, it is likely that this is a global solution. A suggested algorithm for generating alternative starting points is described in reference [4].

4. Subroutines Comprising the Code

The current GRG code is composed of a main program, MAINRG, and 11 subroutines. These are described briefly in the following subroutine dictionary.

Subroutine Dictionary

Subroutine Name	Function	Calls	Called by
1. MAINRG	Not a subroutine. Reads, edits, and prints input.	GRG SUMRY	----
2. SUMRY	Optional user supplied subroutine which prints additional solution output, beyond that provided by GRG	---	MAINRG
3. GCOMP	User supplied subroutine. Given current X vector, computes vector of M+1 function values G, where G(1),..., G(M) are constraint function values and G(M+1) is the objective	---	GRG NEWTON
4. PARSH	Given current X and G vectors, computes array GRAD(M+1, N), whose (I,J) element is the partial derivative of G(I) with respect to X(J). May be user supplied. If not, there is a system subroutine PARSH which computes GRAD by forward difference approximation	If not user supplied, calls GCOMP	CONSBS
5. GRG	Controls main iterative loop. Computes initial BINV and search direction. Calls one dimensional search subroutine DMINRG. Tests for optimality, updates H matrix	GCOMP CONSBS REDGRA DMINRG	MAINRG
6. DMINRG	Performs one dimensional search	REDOBJ CONSBS	GRG
7. REDOBJ	Computes values of basic variables for given values of nonbasics by calling NEWTON. Takes action if NEWTON doesn't converge. Checks for constraint violations. If any are violated, finds feasible point where some initially violated constraint is binding and others satisfied.	NEWTON GCOMP CONSBS PH1OBJ	DMINRG

Subroutine Dictionary - p.2

	Subroutine Name	Function	Calls	Called by
8.	NEWTON	Uses Newtons Method to compute values of basic variables for given values of nonbasics. If convergence not achieved, sets flag and returns	GCOMP	REDOBJ
9.	CONSBS	Computes Basis Inverse, BINV	PARSH	GRG REDOBJ DMINRG
10.	REDGRA	Given BINV and GRAD, computes Lagrange multiplier vector U, and reduced gradient of either phase I or phase II objectives, GRADF	---	GRG
11.	PH1OBJ	If any constraints are violated, computes phase I objective, equal to the sum of constraint violations. Stores this as G(M+1), stores original G(M+1) as TRUOBJ.	---	REDOBJ
12.	TANG	Computes tangent vector v as $v = -B^{-1} (\partial g/\partial x)d$	---	GRG

5. Subroutine Flow Charts and Descriptions

The flow charts in this section are in aggregated rather than detailed form. Their purpose is to describe overall program logic. However, they correspond fairly closely to the actual FORTRAN code. In particular, all array, variable, and subroutine names used are the same as in the code.

Subroutine GRG

The subroutine begins by calling CONSBS to invert the initial basis. If NCAND is not zero in block 1, the user has specified an initial candidate list for CONSBS. Otherwise, all variables are candidates. The arrays IABOVE and IBELOW in block 2 are the sets of indices of constraints which violate their upper and lower bounds respectively.

The Broyden - Fletcher - Shanno (BFS) variable metric algorithm [8] is used to generate the search direction d. This method uses an N x N matrix, H. In block 3, H is initialized to a diagonal matrix with diagonal element zero if the i^{th} nonbasic variable is at a bound and unity otherwise. The test in block 4 is true if either of two optimality tests are passed. The first test checks if the following conditions are met;

for i = 1,N but x_i not a slack variable

for an equality constraint

$$x(i) = LB(i) \Rightarrow GRADF(i) \geq -EPSTOP$$

$$x(i) = UB(i) \Rightarrow GRADF(i) \leq EPSTOP$$

$$LB(i) < x(i) < UB(i) \Rightarrow |GRADF(i)| < EPSTOP$$

The quantities x_i are the current nonbasic variables and GRADF(i) is the i^{th} component of the reduced gradient, (see section 2). This tests whether the Kuhn-Tucker optimality conditions [7] are satisfied to within EPSTOP, a small positive number which can be controlled by the user. The slack variables for equality constraints (i.e. the variables X(N+1) to X(N+NEQ)) are excluded from the test because they must be zero in any feasible solution.

The second optimality test checks if the condition

ABS(FM - OBJTST) < EPSTOP * ABS(OBJTST)

is satisfied for NSTOP consecutive iterations. In the above, FM is the current objective value and OBJTST is the objective value at the start of the previous one dimensional search.

There are two tests for resetting H in block 5. The first tests whether the scalar product

$$\sum_i d(i) * GRADF(i)$$

is negative. If not, the search direction d(i) will not yield an immediate decrease in the objective, and H must be reset. This condition can occur due to numerical error in computing H or to inaccuracies in the one dimensional search. The second test checks if

$$\max |d(i)| < 10^{-6}$$

i.e if d is too small. Neither of the latter two tests can be true immediately after H is reset.

In block 6, ALFMAX is the largest value of α for which each component of $x + \alpha d$ satisfies its bounds. The tangent vector, V, in block 7 is used to compute initial values for the basic variables in subroutine REDOBJ.

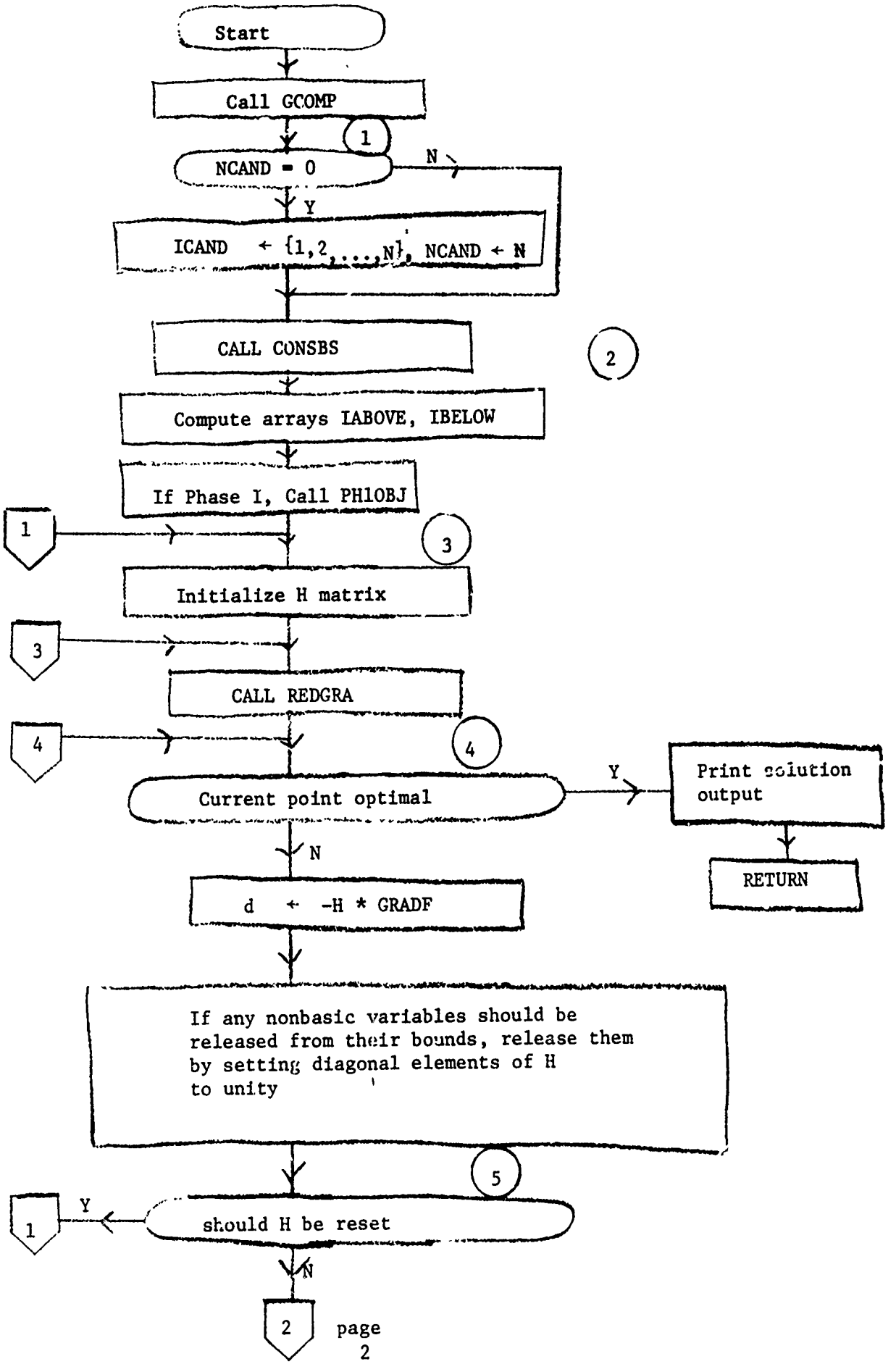
The variable IFLAG in block 8 is set to 3 either in REDOBJ, if phase I ends, or in DMINRG, if a new binding constraint is to be added to the basis.

In either case a new reduced problem is to be solved, so H is reinitialized and the entire procedure begins again. IFLAG is set to 6 either in DMINRG, if too many NEWTON iterations have been taken, or in REDOBJ, if NEWTON fails to converge. A new one dimensional search is initiated but the reduced problem remains the same, so H is not reset.

In block 10, page 2, the binding constraints are checked to see if any have become strictly satisfied during the one dimensional search. If so, a new, smaller basis inverse is constructed in CONSBS, and a new reduced problem solution begins.

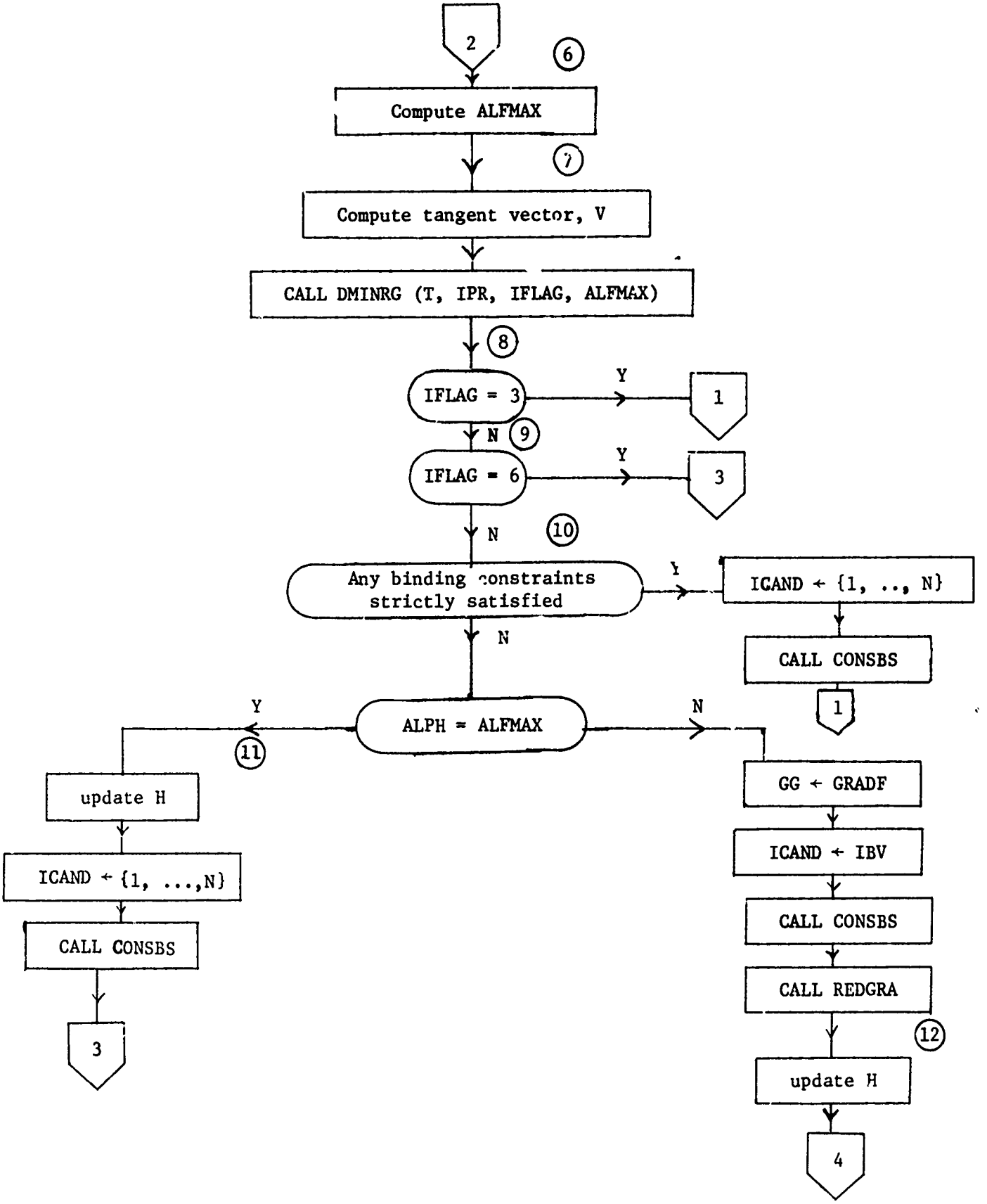
If the one dimensional search ends with IFLAG = 0, corresponding to an unconstrained minimum being located along the search direction d, H is updated and a new iteration begins. The update used depends on whether or not any nonbasic variable has reached a bound, i.e. if the step size, ALPH, is equal to ALFMAX. The updating formulas in blocks 11 and 12 are given in reference [5]. Since H is a symmetric matrix, only the diagonal and super-diagonal elements are needed, and these are stored in a linear array H in row order.

Subroutine GRG - p. 1



page 2

Subroutine GRG - p. 2



Subroutine DMINRG

Subroutine GRG provides search directions for the one dimensional search subroutine, DMINRG, in which the variables of the problem are assigned new values. This subroutine finds a first local minimum for the problem

$$\underset{\alpha}{\text{minimize}} \quad F(\bar{x} + \alpha d)$$

The direction d is always a direction of descent, i.e.

$$d^T \nabla F(\bar{x}) < 0$$

This subroutine searches for three α values, A, B, and C, which satisfy

$$0 \leq A < B < C$$

$$F(\bar{x} + Ad) > F(\bar{x} + Bd)$$

and

$$F(\bar{x} + Cd) > F(\bar{x} + Bd)$$

Then the interval [A,C] contains a local minimum of $F(\bar{x} + \alpha d)$. In block 11 of the flow chart, a quadratic in α is passed through A, B, and C, with its minimum at D. The point D is taken as an estimate of the optimal α and a return is made.

In finding (A,B,C) the choice of initial step size, α_o, (block 1, page 1), is important. With Goldfarbs algorithm or other variable metric methods, α_o is set equal to the optimal α value from the previous search except when this causes too large a change in the variables. The theoretical basis for this is that, as a variable metric converges, the optimal α values should converge to 1, the optimal step size for Newton's Method. Hence the previous optimal step is a good approximation to the current one. This must be modified when the method is restarted, for example when a new constraint is encountered or the basis is changed, since then an optimal step much less than unity is generally taken. Hence, we require that the change in any nonbasic variable larger than 10^{-3} in absolute value not exceed .05

times its value, while the change in any variable smaller than 10^{-3} in absolute value cannot exceed 0.1. If the largest α value meeting these conditions is α^1, and α_i is the step size found by DMINRG at interation i, then α_o is equal to

$$\alpha_o = \min (\alpha_{i-1}, \alpha^1)$$

if the previous search terminated with an interpolation, and $\alpha_o = \alpha^1$ otherwise.

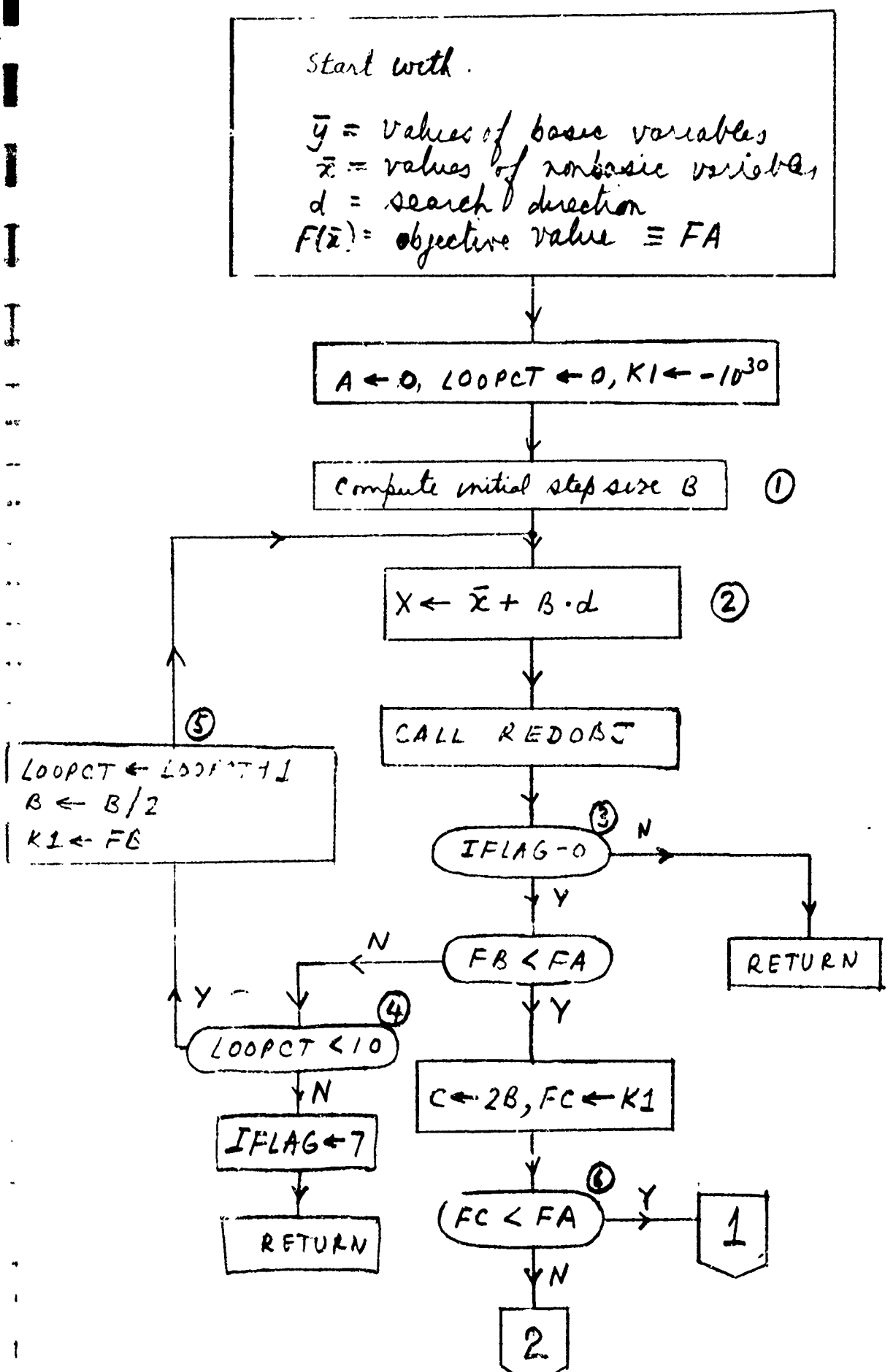
The loop 2 - 3 - 4 - 5 halves the step size until a value FB < FA is achieved, or until LOOPCT = 10. The variable IFLAG in block 3 is set in REDOBJ - to 3 if a new constraint or bound on basic variable was encountered and a new basis was constructed, and to 6 if either NEWTON call in REDOBJ did not converge.

The test in block 6 of the one dimensional search flow chart is false only if the step size has been halved at least once in 2 - 3 - 4 - 5, in which case K1 is the function value corresponding to C. It also insures that the subroutine will cut back. the subroutine will cut back the step size if a large function value is returned by REDOBJ. This is used to force a cutback when the Newton algorithm in REDOBJ does not converge and an improved point has not been found, by setting FB to 10^{30}.

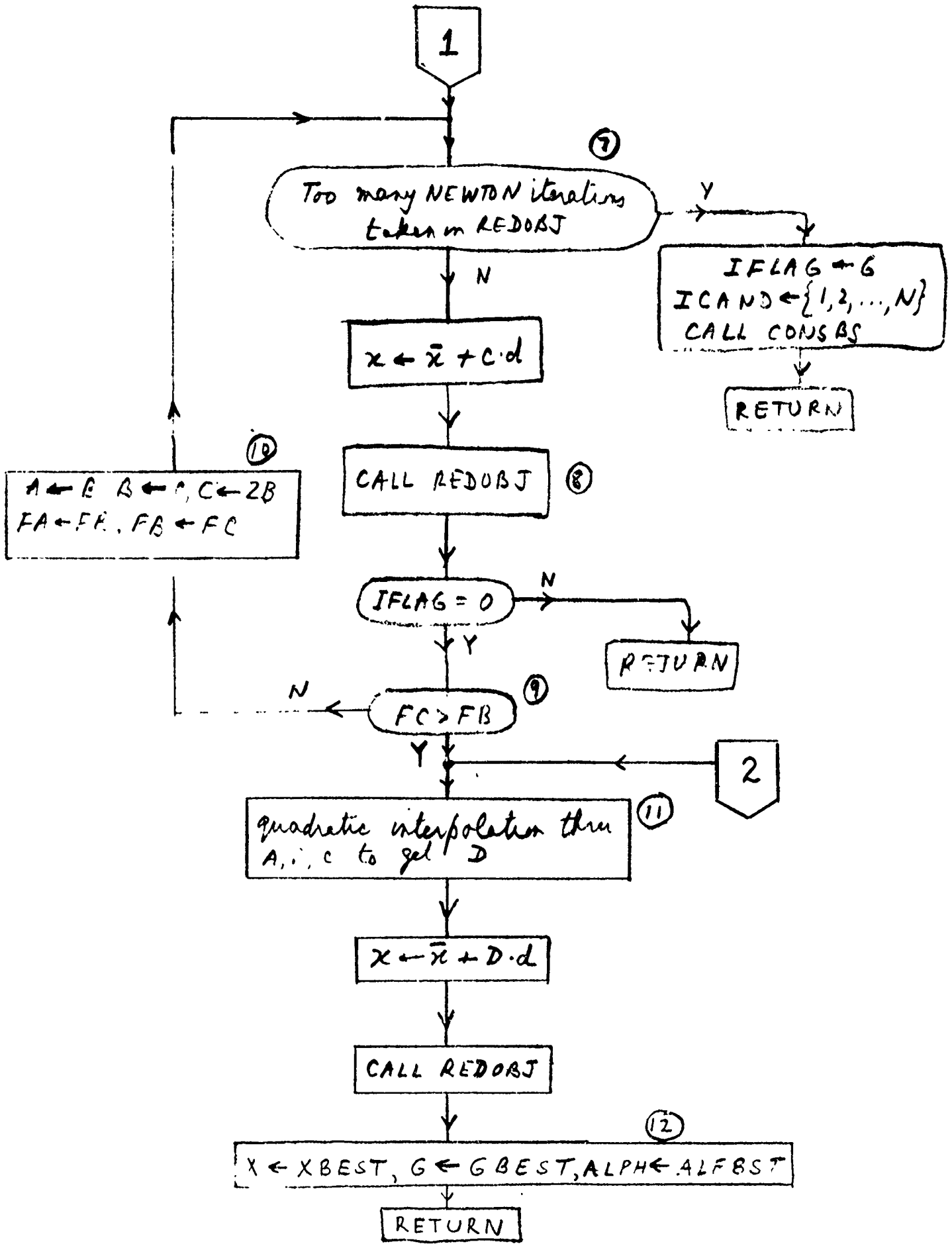
The loop 7 - 8 - 9 - 10 doubles the step size each time until the points A, B, C bracket the minimum. The test in block 7 is true if the NEWTON algorithm in REDOBJ took more than 5 iterations to converge. Experience has shown that, in this case, the next step with C ← 2B is almost certain not to converge in the limit of 10 iterations. This test, and related logic, has reduced overall computational effort significantly - see sections 6 and 7.

Subroutine DMINRG also includes logic to insure that the step taken is no larger than ALFMAX. To simplify the exposition, this logic has not been included in the flow chart. Before returning, DMINRG picks up the best objective value encountered during the search. This is done in block 12, page 2. The quantities XBEST, GBEST, ALFBST are computed in subroutine REDOBJ, which will now be described.

Subroutine DMINRG - P 1.



Subroutine DMINRG - P. 2.



Subroutine REDOBJ - Introduction

This subroutine (REDuced OBJective) is equivalent to the subroutine which evaluates the objective function in a procedure for unconstrained minimization. However, it is much more complicated. REDOBJ is called from the one dimensional search subroutine DMINRG. Prior to the call to REDOBJ, DMINRG chooses a value for the step size α, and computes new values of the non-basic variables, x, equal to $\bar{x} + \alpha d$ (d is the search direction). Then REDOBJ is called. It attempts to compute the corresponding values of the basic variables, by solving the system of NB nonlinear equations

$$g_i\ (y, \bar{x} + \alpha d) = 0 \qquad i \varepsilon IBC$$

for the NB basic variables y, where IBC is the index set of binding constraints. This system is solved by subroutine NEWTON.

If a solution is obtained then all of the constraints are checked to see that none are violated. If not, and if no new constraints are binding then the current objective value is compared to the previous best value. If it is lower, the current values of the variables, constraints, objective and step size are stored as XBEST, GBEST and ALFBST before returning to the calling subroutine (DMINRG).

If the pseudo-Newton algorithm (NEWTON), used to solve for the basic variables in terms of the nonbasics, does not converge then one of two alternatives is taken. If at least one improved point had been found during the linear search process in DMINRG, then the best such value found is accepted as the optimum in this search direction and the search terminated. If no improved point had been found (i.e. ALFBST = 0) then the objective function is assigned a large value ($G(M + 1) = 10^{30}$) to force the linear search process to cut back the step size ALPH and to try again.

If, after the NEWTON process has converged, one of the constraints is

violated then REDOBJ attempts to find the largest value of ALPH such that no constraints are violated and at least one new constraint is binding. If successful, control is returned to DMINRG where a new search is initiated. If not successful then the same action is taken as if NEWTON did not converge.

REDOBJ - Detailed Description

Given the new values for the nonbasic variables x in terms of their previous values $\bar{x}$, the current search direction d and a step size α then the new values for the basic variables y are determined by solving the system of non-linear equations

$$g_i\ (y,\ \bar{x} + \alpha d) = 0 \qquad i \epsilon IBC$$

where IBC is the index set of binding constraints. As in [2] and [3] this is accomplished, in subroutine NEWTON, using the pseudo-Newton algorithm

$$y_{t+1} = y_t - B^{-1}\ (\bar{X})\ g_B(y_t,\ \bar{x} + \alpha d) \quad t = 0,1,2,\ldots$$

where g_B is the vector of binding constraints. The algorithm is called pseudo-Newton because B^{-1} is evaluated once at the initial point of the search, $\bar{X}$, instead of being re-evaluated at each step of the algorithm, as in the standard Newton method.

An initial estimate of the solution is computed by linear extrapolation in block 1 on page 1 of the REDOBJ flow chart. Consider the tangent plane to the constraint surface at $\bar{X}$. This is the set of all vectors (a,b) satisfying

$$(\partial g/\partial y)a + (\partial g/\partial x)\ b = 0$$

where all partial derivative matrices are evaluated at $\bar{X}$. In GRG, the change in x, b, is given by

$$b = \alpha d$$

The corresponding vector a is called the tangent vector, V. Since any scale factor multiplying V is unimportant, we may as well take $\alpha = 1$, yielding

$$V = -\ (\partial g/\partial y)^{-1}\ (\partial g/\partial x)d$$

In our program, V is computed at $\bar{X}$, the initial point of the one dimensional search in block 7, pg. 2, of subroutine GRG. In block 1 of REDOBJ, this vector is used to find initial values, y_o, by the formula

$$y_o = \bar{y} + \alpha_1 V$$

Using these initial values, Newton finds the feasible point X_1. Then, at X_1, V is not recomputed. The old V is used, but emanating now from X_1, to yield the next set of initial values as

$$y_o = y_1 + (\alpha_2 - \alpha_1)V$$

Using these, Newton finds a new point X_2. This procedure is repeated until Newton fails to converge or until the one dimensional search is over.

Newton is considered to have converged if the condition

$$\text{NORMG} = \max_{i \in \text{IBC}} |g_i(X_t)| < \text{EPNEWT}$$

is met within ITLIM interations. Currently EPNEWT = 10^{-4} and ITLIM = 10. If NORMG has not decreased from its previous value (or the above condition is not met in 10 iterations) Newton has not converged. In block 2, ALFBST is the value of α at the best point found thus far in the one-dimensional search. If ALFBST = 0, i.e. no better point has been found, the objective is set to 10^{30} in block 3. This forces DMINRG to cut back the step size. The Newton algorithm will converge if the step size is "small enough". If ALFBST > 0, at least one improved objective value has been found by DMINRG, so CONSBS is called and the search is terminated by setting IFLAG to 6 in block 4.

Once Newton has converged, we check for constraint violations on page 2. There are a number of reasons why the current step α may be too large;

(1) A strictly satisfied constraint (one with index in the set IRC) may have violated an upper or lower bound.

(2) A constraint in lABOVE, the set of constraints initially violating their upper bounds, may violate a lower bound.

(3) A constraint in IBELOW may violate an upper bound.

(4) A basic variable may violate a lower or upper bound.

If one or more of these cases hold, α is reduced to a value, ALFSTR, where no constraints are violated and at least one new constraint is equal to a bound. To determine this constraint, an estimate is made of ALFSTR in block 6 for cases 1, 2, and 3. Linear interpolation between the current and previous values of the violated constraint is used. Case 4 is dealt with in the same way in block 8. If none of the 4 cases holds (Y branch, block 9), the test in block 10 checks if we are still in phase I (NABOVE is the number of indices in IABOVE, the set of constraints violating upper bounds, and similarly for NBELOW). If still in phase I (N branch, block 10) block 11 checks to see if all violated constraints are satisfied. If so, phase I ends and a return is made.

If any of the 4 cases holds, the logic at the top of page 3 decides whether case 4 or one of cases 1 - 3 is to be dealt with. Assuming cases 1 - 3 as an example, we then wish to solve the system

$$g_i\ (y(\alpha),\ \bar{x} + \alpha d) = 0,\ i \varepsilon IBC$$

$$g_L\ (y(\alpha),\ \bar{x} + \alpha d) = 0$$

The Jacobian for this system is

$$J = \left[\begin{array}{c|c} B & c \\ \hline D4 & w \end{array}\right]$$

where

$$D4 = \partial g_L / \partial y$$

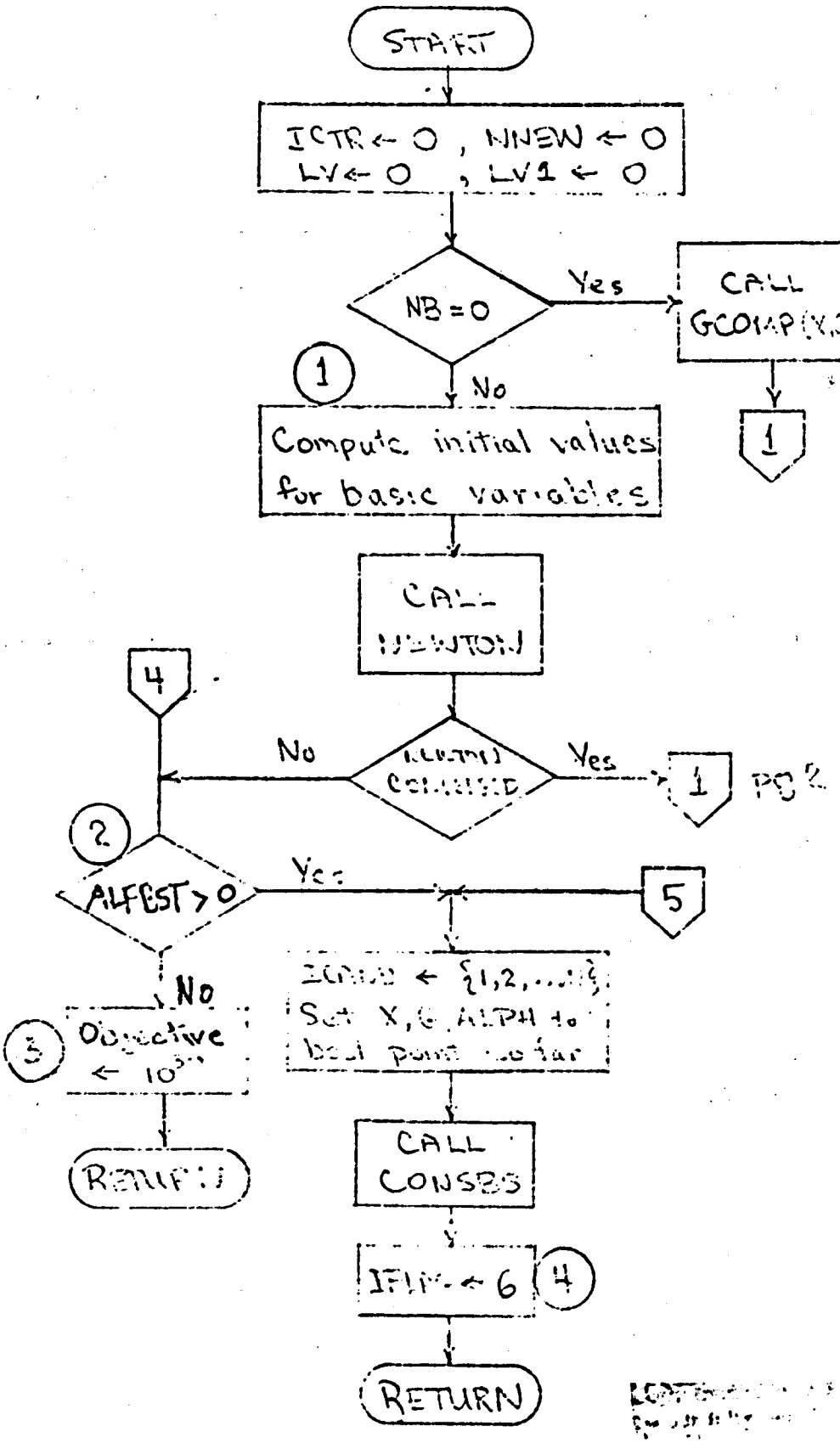
$$w = (\partial g_L / \partial x)^T d$$

and c is an NB component column vector whose elements are $(\partial g_i / \partial x)^T d$ for $i \varepsilon IBC$. Since J involves only adding a border to the current basis matrix,

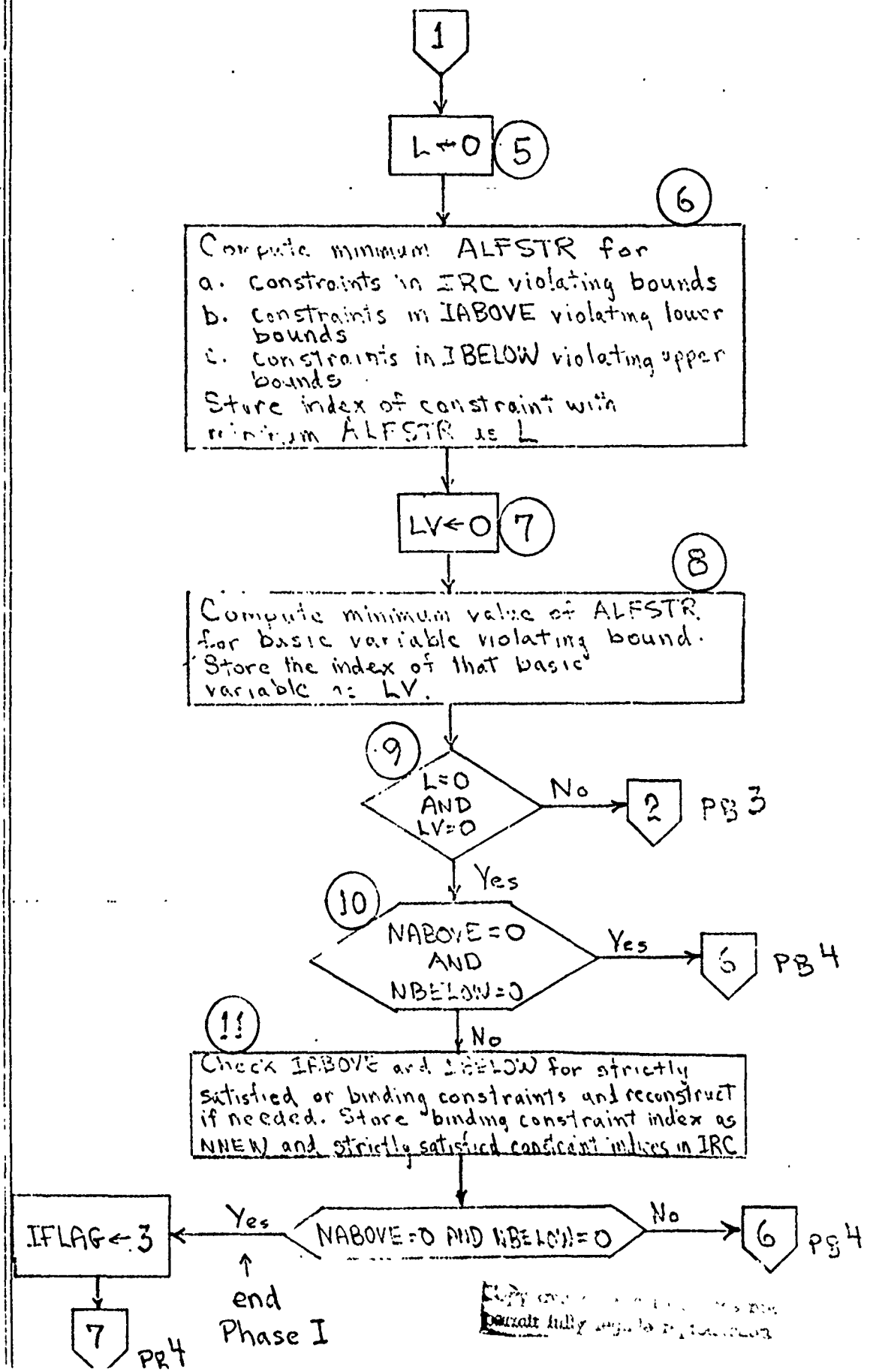
B, its inverse is easily computed if B^{-1} is known. This is done in block 13. The call to NEWTON in block 14 then attempts to solve the system. If NEWTON fails to converge, the same logic as described previously is applied. If it does converge, we return to page 2 to see if any of cases 1 - 4 still holds.

This procedure for satisfying violated constraints terminates with the Y branch in block 9. If a basic variable has been set to one of its bounds, its index is stored as LV1 in block 12, p. 3. In block 15, page 4 LV is replaced by LV1. If both LV and NNEW are zero, the set of strictly satisfied constraints is checked to see if any constraints in it are binding. Then the best point is updated if necessary and a return is made.

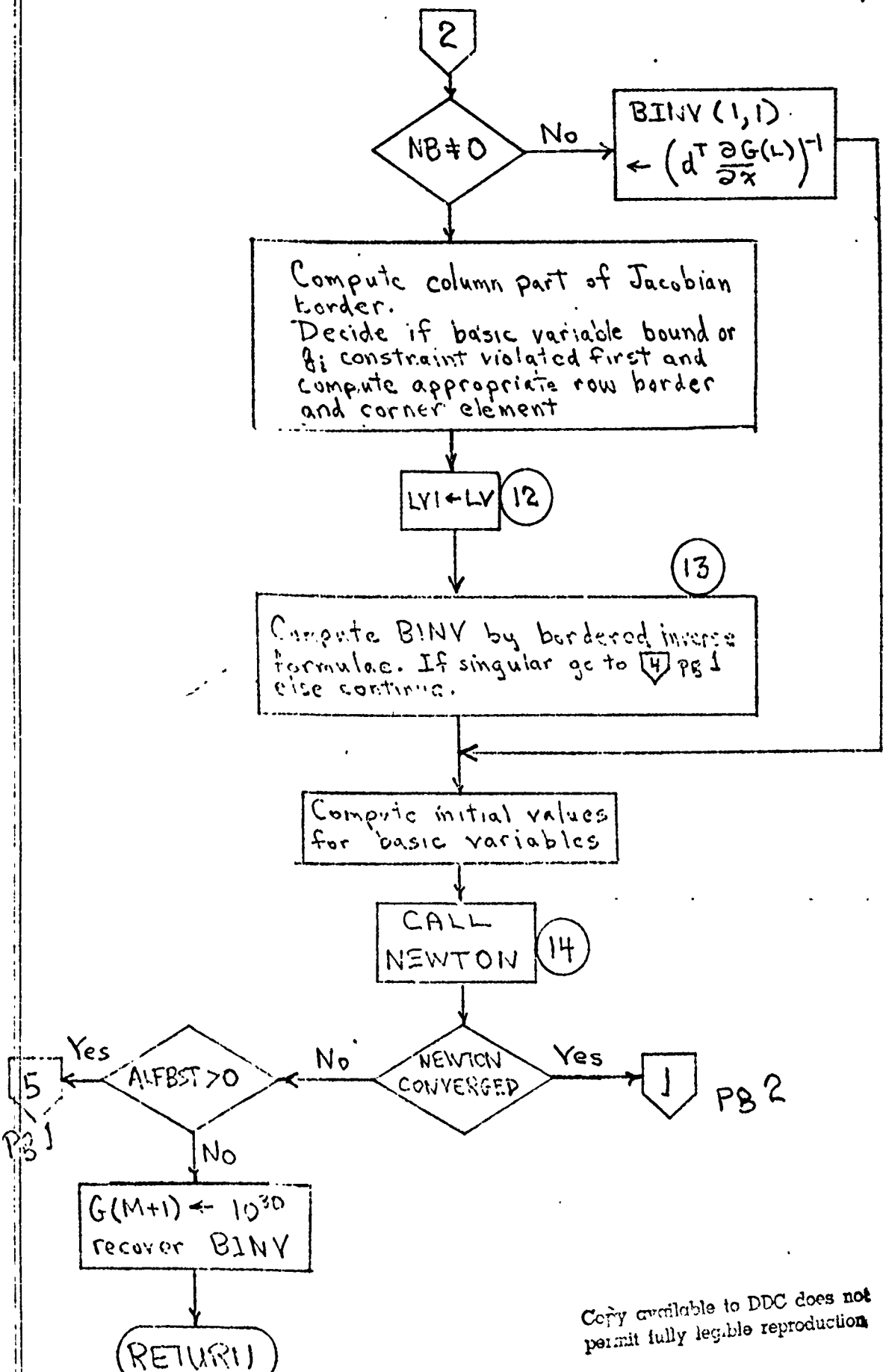
SUBROUTINE REDOK1 PG 1



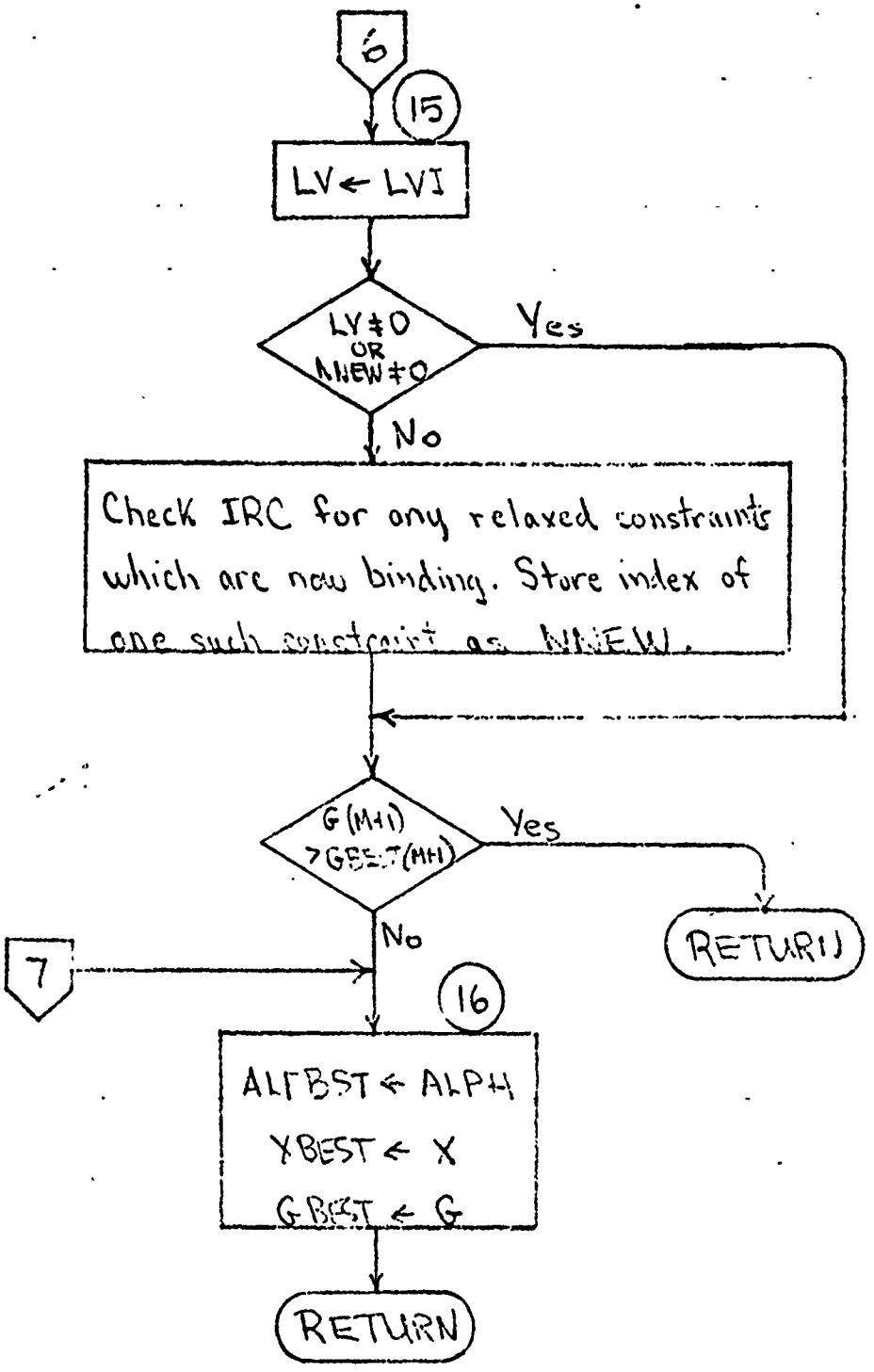
Subroutine REDOBJ



Subroutine REDOBJ. PB 3



Subroutine REDOBJ



Subroutine CONSBS

This subroutine selects a set of basic variables and computes the basis inverse, BINV. Its input is a list of indices of variables, the candidate list ICAND. The outputs of CONSBS are (a) a new list of binding and strictly satisfied constraint indices (b) a new list of basic variable indices and (c) the new basis inverse, BINV. In block 1 of the flow chart the array IREM contains the list of rows of TAB which remain to be pivoted in. The subroutine operates in 2 modes, indicated by the variable MODE. When MODE = 1, CONSBS will choose pivot columns from whatever candidate list was input to it. If a basis inverse could not be constructed from columns in this candidate list, or if the original candidate list included all variables (NCAND = N, block 2), MODE is set to 2, and CONSBS will choose pivot columns from the list of all admissible columns. A column is admissible if it is not scheduled to leave the basis and if it has not yet been pivoted in.

The main loop of CONSBS begins at block 3. A pivot row is chosen as in block 4. The choice of ISV in block 5 is motivated by the desire to have basic variables as far from their bounds as possible, so that fewer basis changes will be required. The other criterion influencing the choice of basic variables is that the basis matrix should be well-conditioned. We try to insure this by choosing as a prospective pivot column that index, I, in ISV yielding

$$\max_{I \in ISV} |TAB(IROW, I)|$$

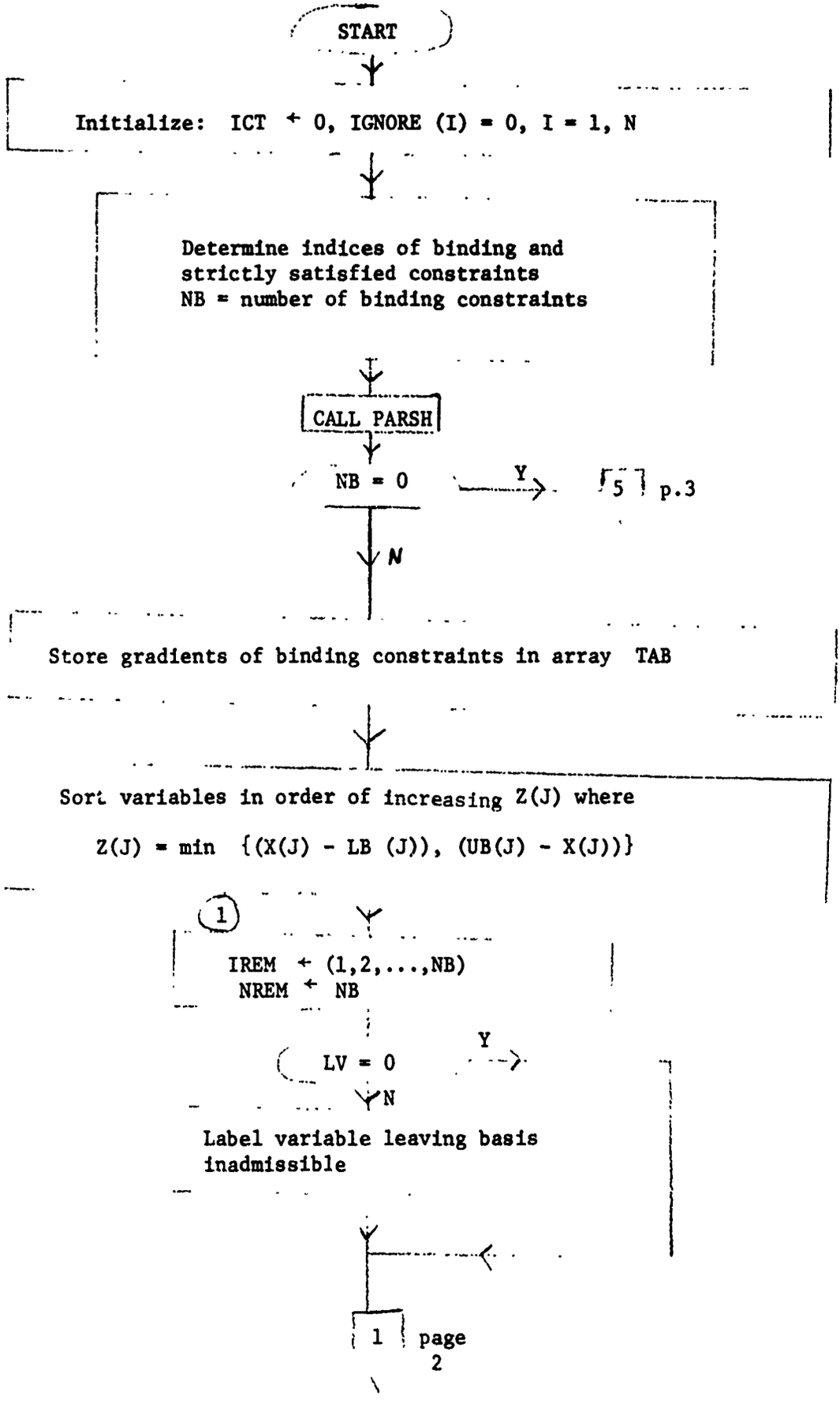
This is done in block 6. If the element chosen passes 2 tests we pivot on it (block 8), transforming the Jacobian and entering that column into BINV. The index of the column pivoted in is stored in the basic variable list, IBV (block 7), and the

procedure is repeated for each binding constraint until either BINV has been constructed (N branch, block 9) or the candidate list has been exhausted (Y branch, block 10, page 2).

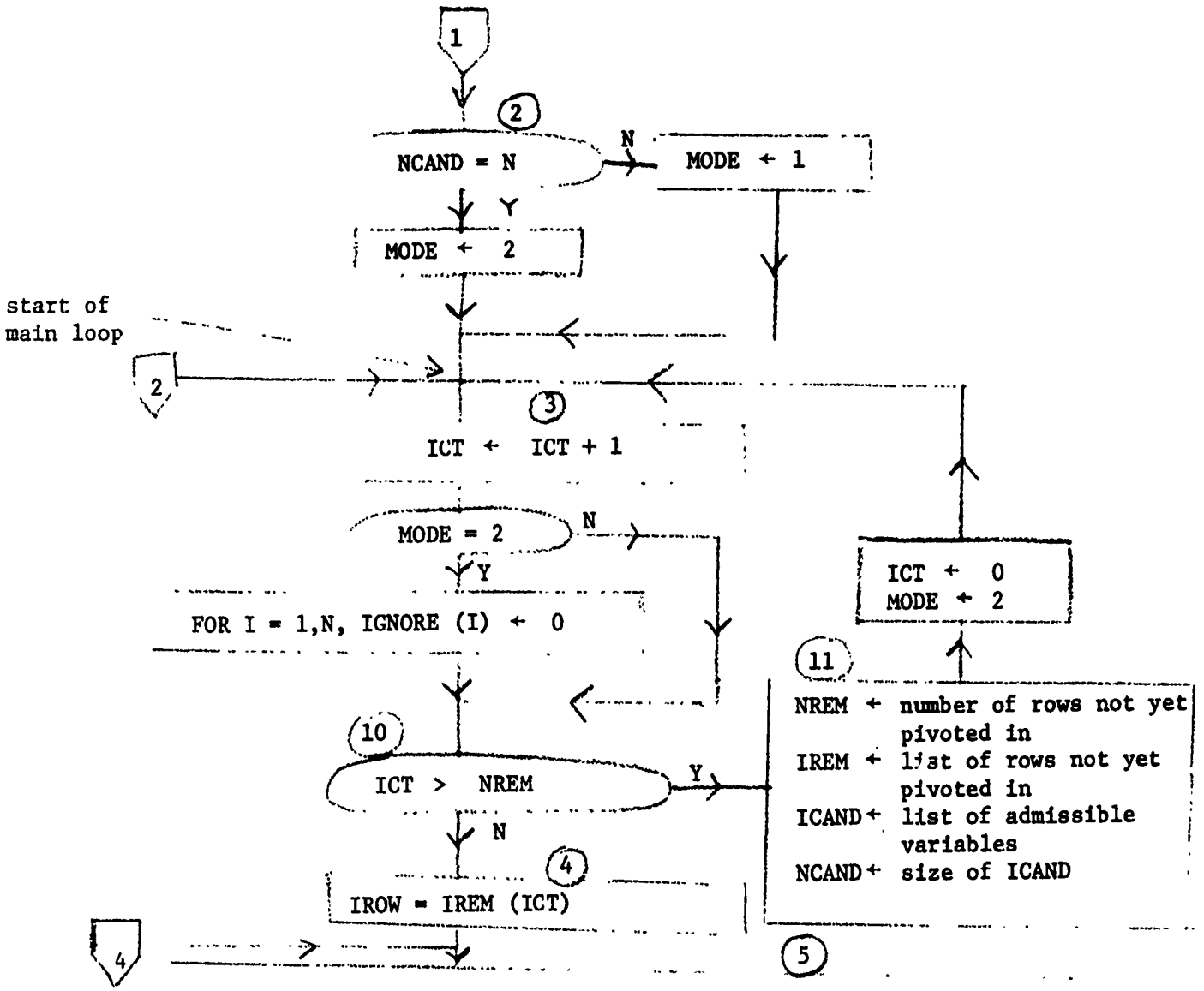
The two tests that a pivot element must pass are (a) its absolute value must be larger than EPSPIV (currently 10^{-6}) and (b) the absolute value of the ratio of all other elements in the pivot column to the pivot element must be less than RTOL (currently 100). The first test insures that we do not pivot on an element that is essentially zero, while the second protects against the generation of elements of large absolute value in BINV. Such values are symptomatic of ill-conditioned basis matrices. It either test is failed while MODE = 1, we simply do not pivot in the current row, and move on to the next one.

If, when mode 1 terminates, BINV has not yet been constructed, we attempt to complete its construction by considering columns not in the originalcandidate list. In block 11, page 2, the candidate list, ICAND, is reset to the set of all remaining admissible columns, MODE is set to 2, and we return to the start of the main iterative loop. If, in this second phase, a pivot element fails the absolute value test, we temporarily mark all columns in ISV inadmissible (by setting their indicator in the IGNORE array to 1, in block 12, page 3) and choose a new ISV array. If, in mode 2, a column fails the RTOL test (block 14, page 3), its IGNORE indicator is set to one. If all admissibl matrix elements in a row fail one of these two tests (N branch, block 13, page 2) we are forced to consider variables within EPBOUN of their bounds. This is done in block 15, p.4. If all these fail the tests, the slack variable corresponding to row IROW is entered into the basis.

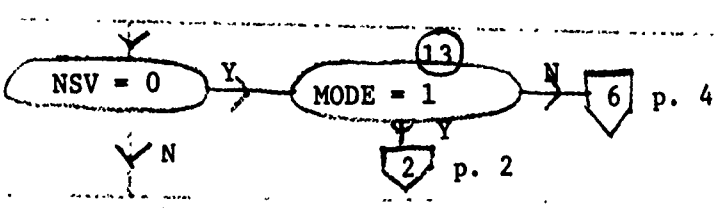
Subroutine CONSBS - p. 1



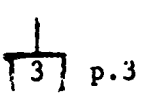
Subroutine CONSBS - p.2



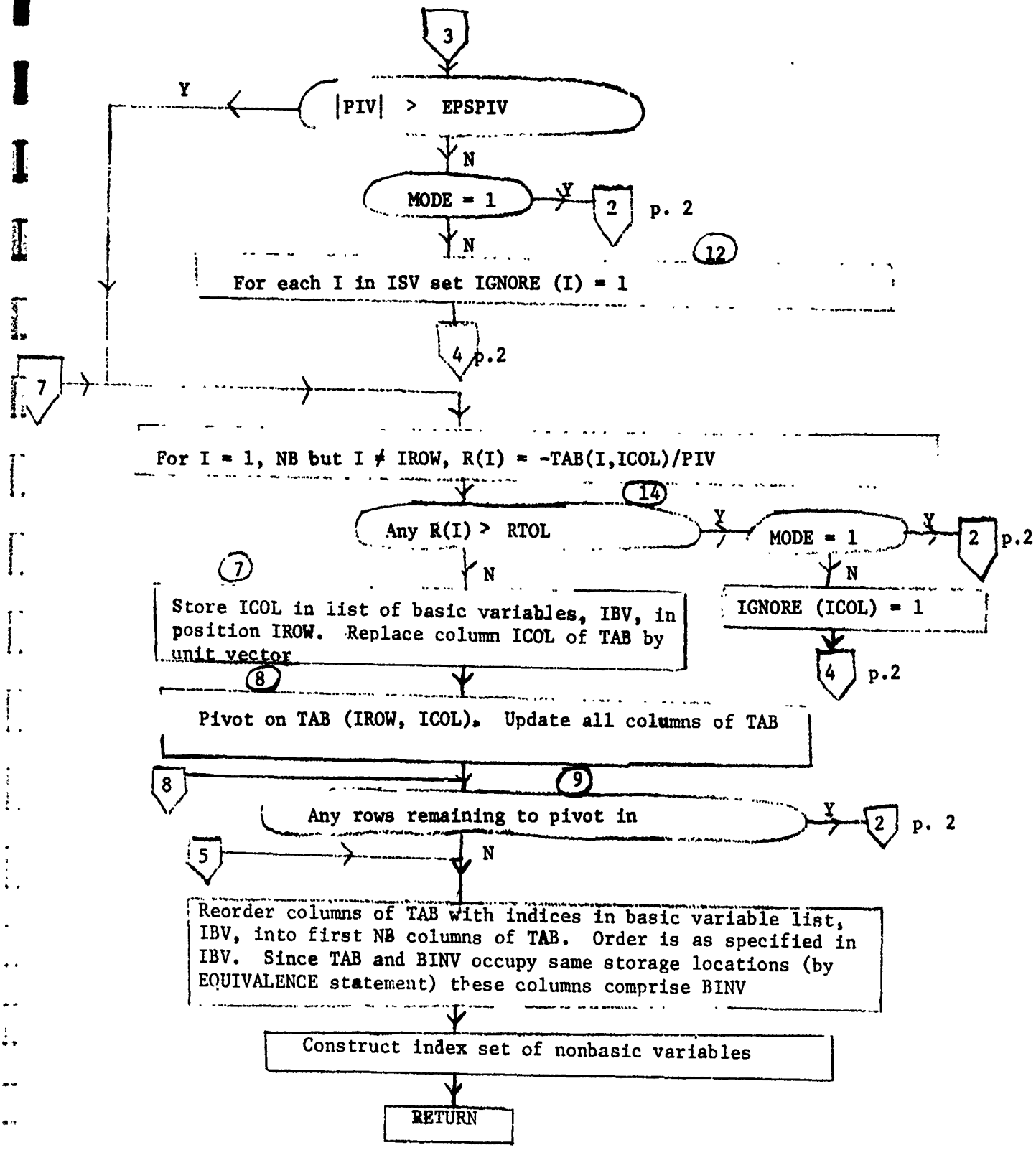
Select (up to) 5 admissible candidate variables with IGNORE (I) = 0 which have Z(J) > EPBOUN and with largest values of Z(J). Store indices in ISV. Let NSV = number of indices in ISV



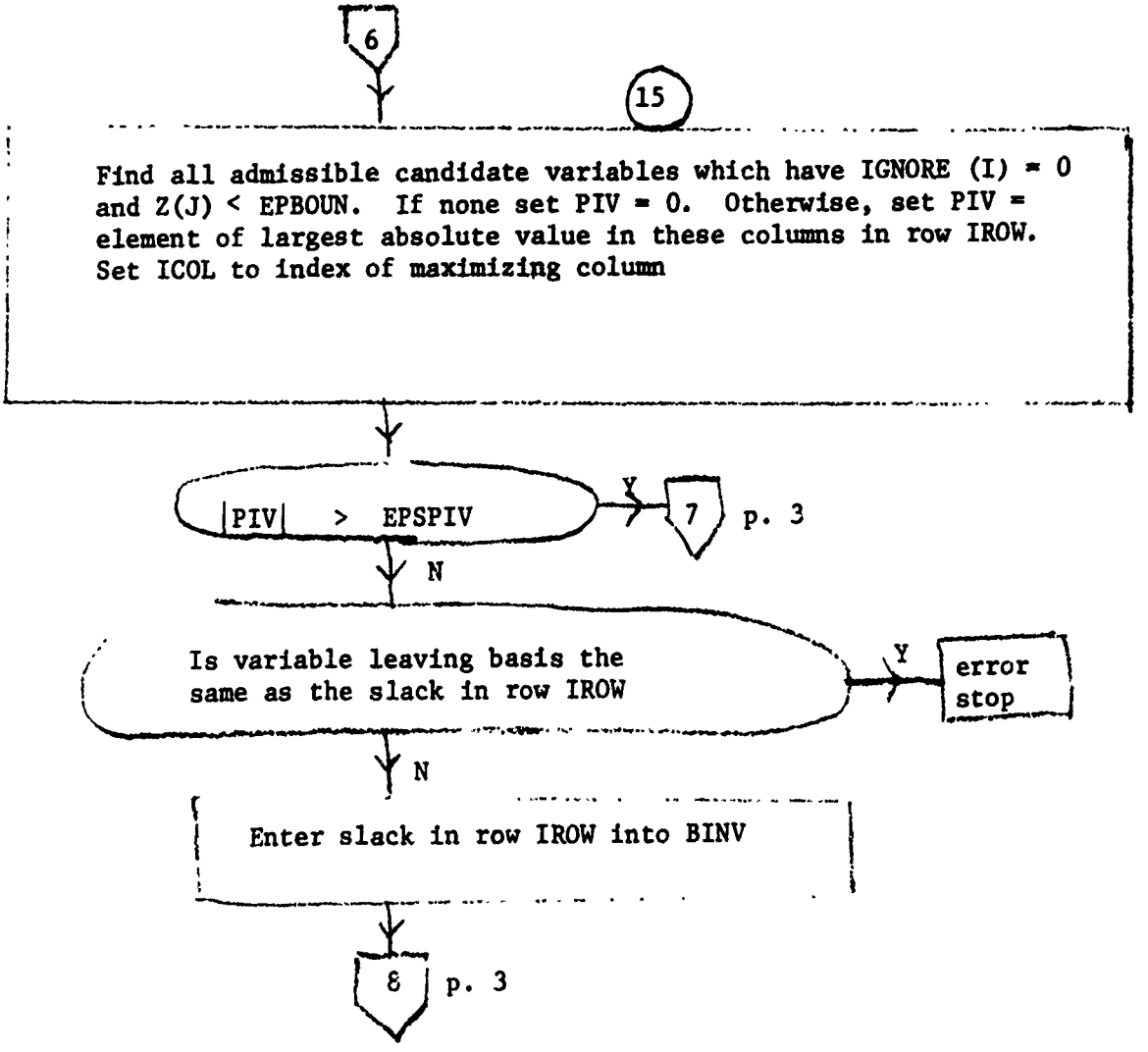
Scan row IROW of TAB. From columns with indices in ISV, pick one with element of largest absolute value. Let element value = PIV, column index = ICOL



Subroutine CONSBS - p. 3



Subroutine CONSBS - p. 4



6. Changes in the Algorithm

The subroutines of section 5 have been changed significantly from their orginal versions, described in reference [1]. Here we discuss some of these changes and give reasons for them.

Subroutine PARSH

A finite difference version of PARSH has been added. This computes partial derivatives of $g_1,..,g_{M+1}$ by simple forward differencing, using a constant increment of 10^{-4} in each variable. It frees the user of having to code his own PARSH, a task which can require man - months of effort for some problems. In comparing solution by using analytic and finite difference derivatives, little or no degradation in accuracy or speed has been noted.

The form of the user - supplied PARSH has also been changed. Previously, the code assumed that only the gradients of currently binding constraints would be computed by PARSH. Now, gradients of all constraints are computed. This significantly simplifies the preparation of PARSH. The older version required that PARSH compute the gradients of an arbitrary subset of constraints, which is more complex to code than simply computing all of them. The rest of the program is also simplified. No distinction need now be made between finite difference and analytic derivatives; previously, finite difference would compute all gradients, analytic only those for binding constraints.

Subroutine GRG

The Broyden - Fletcher - Shanno (BFS) variable metric algorithm [8] has replaced the Davidon - Fletcher - Powell [6] method. This was done because recent computational experience indicates that BFS is the best of the variable metric algorithms. Reference [9] shows that BFS is less sensitive to errors in the 1 dimensional search than DFP, while [10] provides evidence that periodic restarting of BFS is undesirable. Hence a simpler 1 dimensional search is now used, and the test in block 5 of GRG does not include resetting H periodically.

Subroutine DMINRG

The cubic interpolation section has been deleted. This followed the quadratic interpolation (block 11), and fit successive cubics through 4 points until certain stop criteria were met. It was removed because:

(1) In about 10 test problems, it was found that deleting the cubic interpolatio increased solution effort little, if at all, and often decreased it. There are a number of reasons for this. Foremost among them is the fact that, to compute the reduced objective F(x) exactly, the basic variables y(x) must be determined exactly, i.e. each binding constraint must be exactly equal to zero. Of course this is impossible in practice, and the binding constraints are only within EPNEWT of their bounds, with the default value of EPNEWT currently equal to 10^{-4}. Hence F(x) is computed significantly less accurately than if the problem were unconstrained, perhaps only to 4 or 5 significant figures. With such noise in the function evaluation, cubic interpolation rarely achieves improved functio.n values.

(2) There is much evidence in unconstrained minimization that a "sloppy" one dimensional search achieves better overall results than a more exact one.

(3) The code was significantly simplified and shortened.

The current interpolation strategy is simply to bracket the minimum, fit a quadratic, choose the best of all points evaluated during the current search, and return. Multiple interpolations are no longer performed. Choosing the best point is necessary, because the D point selected by the quadratic interpolation (block 11, DMINRG flow chart) may not be the best, and choosing it can yield a final objective value worse than the value at the start of the search.

Two new checks have been added to the quadratic interpolation. The first test requires that the product (A-B) * (B-C) * (C-A) exceed, in absolute value, a tolerence EPSQ1, while the second requires that the three points (A,FA), (B,FB) and (C,FC) enclose an area no less than a tolerence EPSQ2. These tests ensure that the quadratic interpolation is numerically stable, and no interpolation is performed if either of th fails.

Subroutine REDOBJ

The actions taken when NEWTON does not converge (see pages 1 and 3 of the flow chart) are new. In the current logic, if NEWTON does not converge then either the best point found so far is accepted as the minimum and the one dimensional search terminated or, if no such point has been found, the objective value at the current point is set to 10^{30} in order to force the step size to be cut back in DMINRG. Previous logic tried to force convergence by first computing an approximate B^{-1} and, if that also failed, then an exact B^{-1}, both evaluated at the last feasible point. The approximate B^{-1} was a dismal failure, and even the exact one often would not cause Newton to converge. Further, when convergence of Newton did ensue, many iterations were usually required and, of course, evaluating B^{-1} involves a significant amount of computation. Computational results (see section 7) show a significant improvement with the new logic. Evidently, once the radius of convergence of Newtons method (with B^{-1} evaluated at $\alpha = 0$) has been reached, it is not worthwhile to try to extend it. In fact, it had proved worthwhile to limit motion from $\alpha = 0$ to those α values which require only a few Newton iterations (fewer than 6, for example) for convergence. This is accomplished by the test in block 7, page 2 of the DMINRG flow chart.

Additional advantages of this new logic are (1) it is much simpler and (2) only one evaluation of B^{-1} is required per one dimensional search. This latter feature is especially important for large problems, where evaluating B^{-1} will be one of the major computational steps.

Subroutine REDOBJ now includes logic to deal with basic variables which violate their bounds. This permitted the elimination of subroutine BSCHNG of reference [1], and hence shorted the code.

The logic in blocks 6 and 11 on page 2 was added as part of the changes required to solve problems for which the initial solution was not feasible i.e. a Phase I to determine an initial feasible solution. These blocks deal with constraints which had previously violated an upper (lower) bound and now, because too large a step has been taken, violate their lower (upper) bounds.

The logic on page 4, block 15, which stores the point with lowest objective value, is also new. This causes each one dimensional search to yield a point no worse than the one it starts with. This logic was added because the last point obtained by DMINRG is not always the best, especially when it is obtained by interpolation. This may be due to the inacurracy with which F(x) is obtained; interpolation seems to produce poorer results as EPNEWT is increased. Without this logic, function values obtained at the end of successive DMINRG calls sometimes increased, especially near the minimum and with larger values of EPNEWT. One consequen of this was that many iterations could take place without the objective improving and without thc stop criteria in subroutine GRG being met. The new logic eliminates this problem.

Subroutine CONSBS

The only major change from the previous version is in the way degeneracy is handle In block 13, p. 2 of the CONSBS flow chart, the N branch is taken when an acceptable pivot element cannot be found in mode 2. The old code simply printed an error message and stopped. The present code branches to page 4 of the flow chart, which enters a variable at its bound into the basis. This is an improvement, but the logic is still not satisfactory. Future plans for dealing with degeneracy are discussed in section 8.

7. Computational Results

(a) Comparison with Previous GRG Code

The changes described in section 6 have both simplified the code and improved its efficiency. Table 1 below gives problem characteristics of eight test problems, while table 2 gives comparative results using the new GRG logic and the old described in reference [1] and section 6. While the number of one dimensional searches has increased slightly, the other 3 performance measures are significantly improved.

Problem No.	NAME OF PROBLEM	NO. OF VARIABLES	NO. OF EQUALITY CONSTRAINTS	NO. OF INEQUALITY CONSTRAINTS
1.	Kowalik-Osborne Quadratic	4	0	3
2.	Colvil. Quadrat.c	5	0	3
3.	Asaadi Problem No. 4	5	3	0
4.	Eight Variable Spring	8	0	23
5.	R.A.C. Shell Dual	15	0	5
6.	Seven Variable Truss	17	10	8
7.	RAC Primal	5	0	10
8.	GGP Alklyation	7	0	14

TABLE 1 - Characteristics of Test Problems

PROBLEM	1		2		3		4		5		6		TOTALS	
STATISTIC	OLD	NEW	OLD	NEW	OLD	NEW	OLD	NEW	OLD	NEW	OLD	NEW	OLD	NEW
One Dimensional Searches	8	8	5	5	6	9	8	12	31	30	10	11	68	75
Newton Iterations	135	56	8	9	519	110	87	48	543	412	87	78	1379	813
Equivalent Function Calls *	283	143	55	48	724	199	234	199	1299	1000	324	208	2874	1797
BINV Computations	15	8	6	5	35	9	15	12	44	30	11	11	126	85

TABLE 2 - Results of New Logic

* Equivalent Function Calls = Functions Calls + N• Gradient Calls
where N = No. of variables.

(b) Comparison with Penalty Methods.

The same seven test problems were run on the current GRG code as well as on the interior penalty code described in [11]. GRG required far fewer on dimensional searches, function evaluations and gradient evaluations. While some of this reduction was offset by the requirement of matrix inversion and solution of nonlinear equations in GRG, it was noted that GRG produced more accurate solutions for most of the problems. Differences in computation times (of the order of hundredths of a second) could not be estimated accurately owing to the masking effects of mutiprogramming. We expect that GRG will prove superior to penalty methods for large problems where linear programming technology can be implemented very effectively. The comparative performance of GRG and the penalty codes is shown in Table 3

STATISTIC	TOTALS - PROBLEMS 1,2,4-8		REDUCTION FACTOR PENALTY/GRG
	PENALTY	GRG	
One Dimensional Searches	495	94	5.3
Function Calls	3773	1125	3.4
Gradient Calls	539	99	5.4
Equivalent Function Calls	8645	2023	4.5
Newton Iterations	-	747	-
BINV Computations	-	94	-

Table 3 - Comparison of GRG and Interior Penalty Codes.

8. Future Work

a. Degeneracy

A feasible point is degenerate if the gradients of the binding constraints (including bounds on the variables) are dependent. Hence, a point with more than N constraints binding is always degenerate. At a degenerate point, any basis matrix must include at least one column corresponding to a variable at one of its bounds. This may be a slack variable or one of the natural variables. As one attempts to move away from a degenerate point, it may be that one or more of the basic variables at a bound immediately violates that bound. Then either the basis must be changed or a new search direction selected. We have encountered 3 degenerate points in the dozen or so problems solved thus far. The current logic for dealing with degeneracy is unsatisfactory. Work in the coming year will aim at remedying these defects.

b. Extrapolation of Basic Variables

Within a given one dimensional search, each basic variable, y_i, is a function of the step size α, $y_i = y_i(\alpha)$. At $\alpha = 0$ we know $y_i(0)$ and $y_i'(0) = V_i$, where V_i is the i^{th} component of the tangent vector V. At $\alpha = B$ we compute $y_i(B)$. Given this information, a quadratic can be fit to $y_i(0)$, $y_i'(0)$, $y_i(B)$ and used to estimate $y_i(C)$. These estimates are used as initial values in the Newton iteration which evaluates $y_i(C)$. Then a quadratic can be fit thru $y_i(0)$, $y_i(B)$, $y_i(C)$. This is used to estimate y_i at the next α value, and so on. At each stage, we have a quadratic approximation to the curve $y(\alpha)$ on the constraint surface. These approximations should be much better than the current linear approximation, which always uses the tangent vector V evaluated at $\alpha = 0$. Computational

experiments to measure the improvement yielded by these moving quadratic extrapolations will be carried out in the coming year.

c. Conjugate Gradient Algorithm

We have changed from the DFP to the BFS variable metric algorithm, due to recent work by Shanno [9, 10,] showing it to be superior. However, all variable metric algorithms require storage and updating of an N by N H matrix which is not sparse. Hence there is no way they can be used in a GRG algorithm that is to solve large problems. The only alternative algorithm that requires no H matrix and converges finitely on quadratic functions is the Conjugate Gradient (CG) algorithm [12]. We plan to implement a version of GRG employing CG and compare its performance with that of the BFS procedure. CG should permit solution of problems with 100 constraints and two to three hundred variables in an explicit inverse GRG code which does not exploit sparsity.

d. Solution of Geometric Programs

Geometric programs (GP) have constraint and objective functions of the form

$$g_k(X) = \sum_{i=n_{k-1}}^{n_k} c_i\, t_i(X)$$

where the terms t_i are given by

$$t_i(X) = \prod_{j=1}^{N} X_j^{a_{ij}}$$

The exponents a_{ij} are arbitrary real numbers. Such problems have received much attention in the literature, and special algorithms based on the dual GP have been developed to solve them. These algorithms are not completely satisfactory, however, due to ill-conditioning

of the dual objective function [14]. We intend to solve a number of geometric programs using GRG and compare the results with those obtained by other researchers with whom we are in correspondence. Our GRG code will be specialized to the GP structure by adding a "front end" module. This will permit input of the GP by specifying the indices n_k, constants c_i, and exponent matrix a_{ij}. It will also compute derivatives of the functions g_k as the terms t_i are evaluated. This will speed computation significantly, and will yield the accuracy of analytic derivatives without any user-supplied PARSH subroutine.

e. A GRG Code for Large Problems

One of the most attractive features of GRG is that it is able to accommodate large problems (1000 or more constraints and many more variables) by incorporation of techniques which exploit sparsity. These techniques are extensively used in modern linear programming codes. A long range (2 - 3 years) goal of this research is the development of a large scale GRG code and its comparison with current codes for solving large NLP's, e.g. approximation programming codes [13].

References

1. L.S. Lasdon, R. Fox and M. Ratner, "Nonlinear Optimization Using the Generalized Reduced Gradient Method" Tech. Memo. No. 325, Department of Operations Research, Case Western Reserve University, October 1973.

2. J. Abadie and J. Carpentier, "Generalization of the Wolfe Reduced Gradient Method to the Case of Nonlinear Constraints" in Optimization, R. Fletcher, Ed., Academic Press, 1969, pp. 37-47

3. J. Abadie, "Application of the GRG Algorithm to Optimal Control Problems" in Nonlinear and Integer Programming, J. Abadie, Ed., North Holland Publishing Co., 1972, pp. 191-211.

4. J. Hartman, "Some Experiments in Global Optimization" Technical Memorandum No. NPS55HH72051A, United States Naval Postgraduate School, Monterey, California, May 1972.

5. D. Goldfarb, "Extension of Davidons Variable Metric Method to Maximization Under Linear Inequality and Equality Constraints", SIAM J. Appl. Math., Vol. 17, No. 4, July 1969.

6. R. Fletcher and M.J.D. Powell, "Rapidly convergent Descent Method for Mimimization", Comp. J., vol. 6, p.163 (1963)

7. L.S. Lasdon, "Optimization Theory for Large Systems", Macmillan, New York, 1970, pp. 73-83

8. R. Fletcher, "A New Approach to Variable Metric Algorithms", Computer Journal, Vol. 13, 1970, pp. 317 - 322

9. D.F. Shanno and K.H. Phua, "Inexact Step Lengths and Quasi Newton Mehtods", working paper, University of Toronto, 1974.

10. D.F. Shanno, A. Berg and G. Cheston, "Restarts and Rotations of Quasi - Newton Methods", in Information Processing 74, North Holland Publishing Co., 1974 pp. 557 - 561.

11. R.L. Fox, L.S. Lasdon, A. Tamir, and M. Ratner, "An Efficient One Dimensional Search Procedure", Management Science, to appear

12. R. Fletcher and C.M. Reeves, "Function Minimization by Conjugate Gradients", British Computer Journal, Vol. 7, 1964, pp.149 - 154.

13. R.E. Griffith and R.A, Stewart, "A Nonlinear Programming Technique for the Optimization of Continuous Processing Systems", Management Science, Vol. 7, 1961 pp. 379 - 392.

14. Kochenberger, G.A, et. al., "On the Solution of Geometric Programs via Separable Programming", Operational Res. Quarterly, 24(2), 1973